Optimal Scheduling Using Model Checking

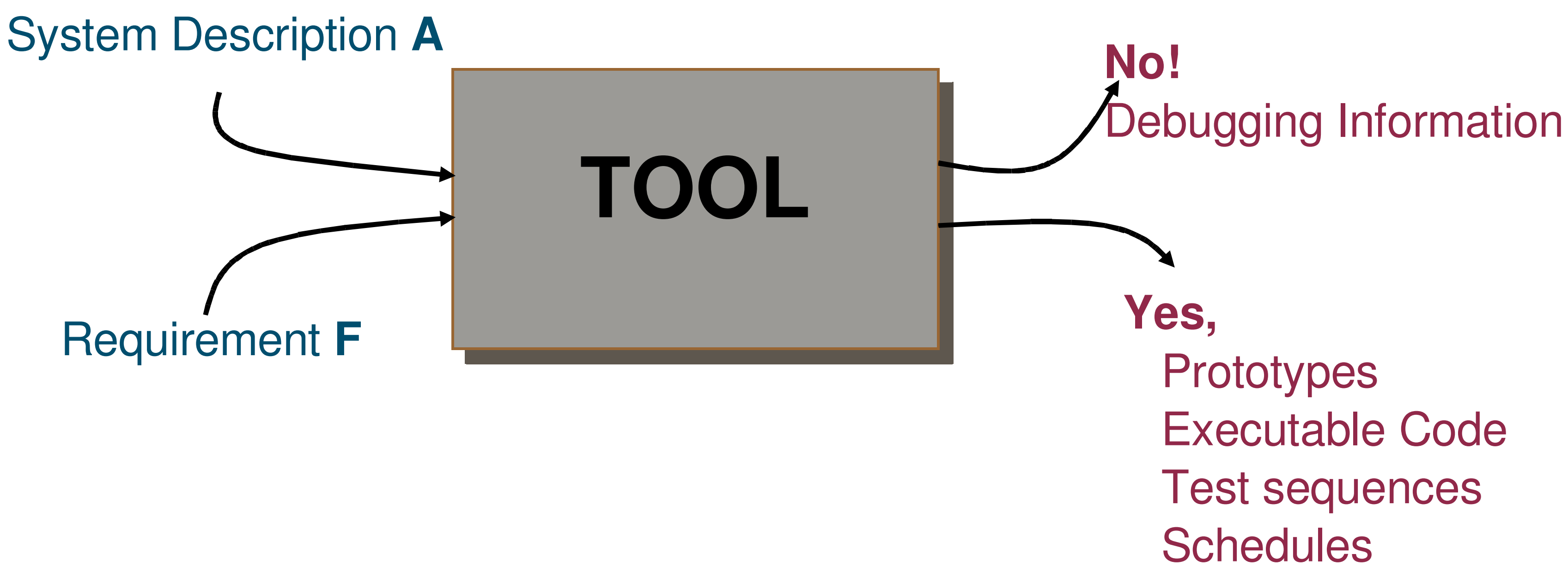
Juhan P. Ernits
Institute of Cybernetics / Tallinn Univ. of Technology

CS theory days, Sept 30, Voore

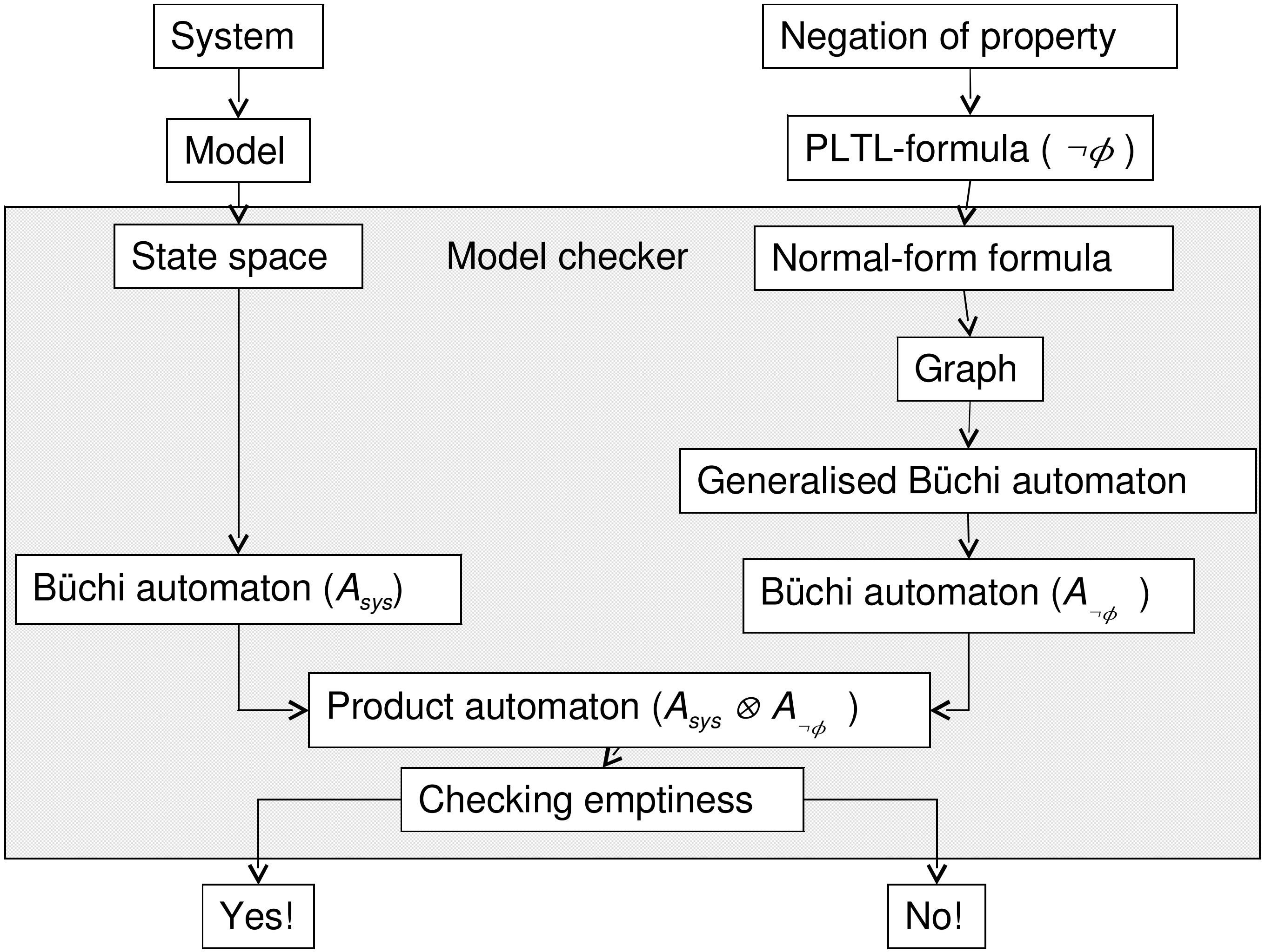
Motivation

- Model checkers are highly optimised search engines
- Any model checker can produce a witness trace
- It seems feasible to describe an optimisation problem as a model (a transition system) and use a model checker to find an answer (the witness trace)

Model checking



Model checking LTL



Optimal Scheduling in Spin

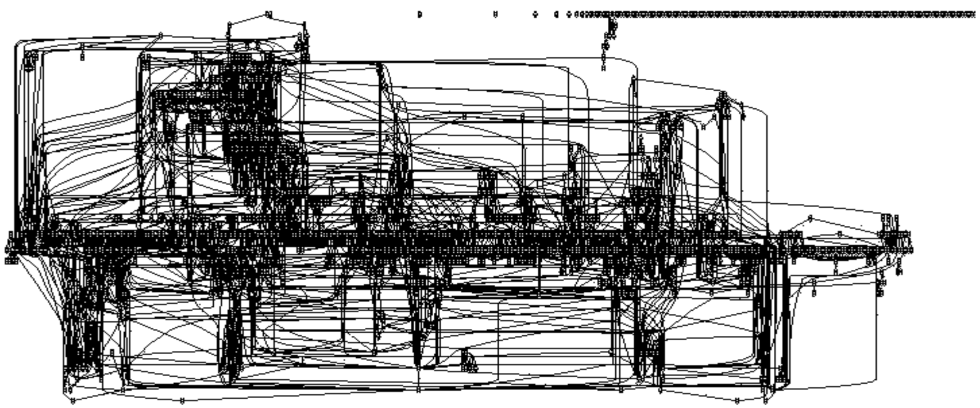
- Optimal scheduling in Spin has been described by Theo Ruys in [1] (the first part of the talk is based on his paper)
- We'll have a look at how
  - scheduling problems can be specified as Promela models
  - to call internal functions of Spin to do branch and bound
- We'll also have a (breef) look how such scheduling can be scaled using bitstate hashing-based iterated search refinement

The last intro slide

- The operational research community has solved many of the standard optimisation problems very efficiently (e.g. the Euclidean travelling salesman)
- Model checking is interesting for optimisation in cases where one needs to add new constraints and modifying the highly optimised algorithms is hard

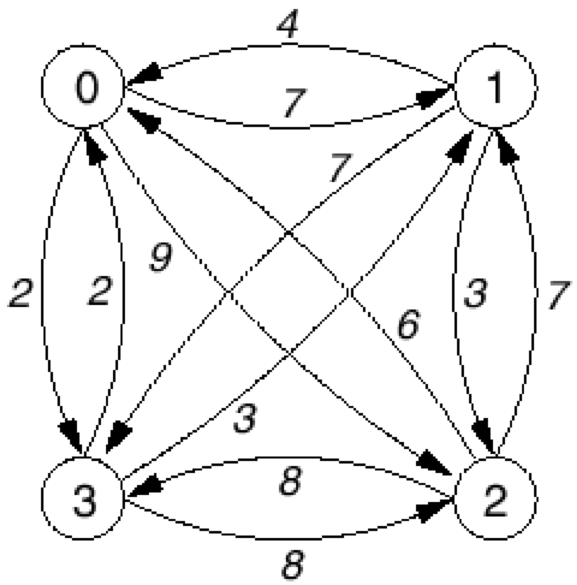
Why Promela?

- A language instead of state machines is useful because this



does not seem to be very clear. Thus textual description of systems seems useful too.

Example: Travelling Salesman



	0	**1**	**2**	**3**
0	-	7	9	2
1	4	-	3	7
2	6	7	-	8
3	2	3	8	-

- Find the shortest path that passes all towns

Promela 101

- Spin models consist of
  - variables
    ```
    bit visited[3];
    int cost;
    ```
  - processes
    ```
    active proctype TSP()
    {...}
    ```
  - message channels

    (we do not use them in the current example)

Promela 101

- Within a process selection can be impemented in the following way:

```
P0: atomic {
        if
        :: !visited[1] -> cost = cost + 7 ;goto P1
        :: !visited[2] -> cost = cost + 9 ; goto P2
        :: !visited[3] -> cost = cost + 2 ; goto P3
        fi ;
}
```

The Specification

- The property can be specified as

```
<> higher_cost
```

where

```
#define higher_cost (c_expr { now.cost >= best_cost })
```

- Notice that the property changes during the search!

Extensions to Promela

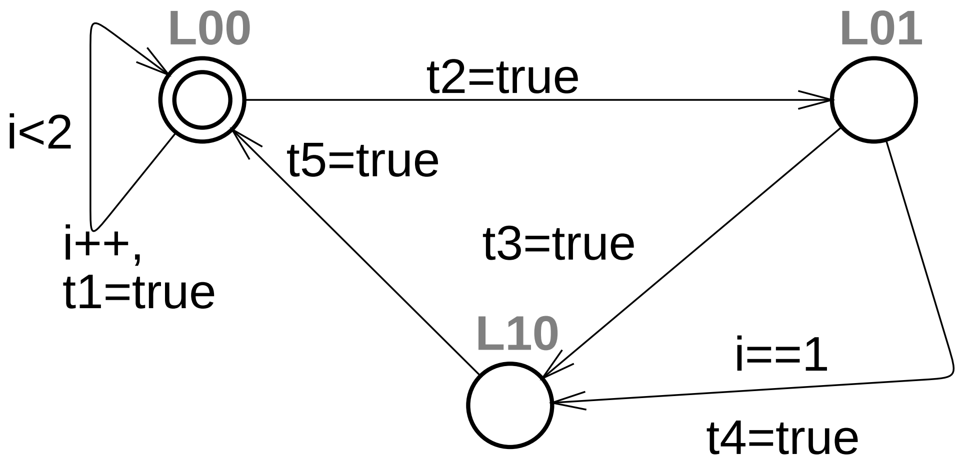
- c_decl - introduce C types that can be used in the Promela model
- c_state - add new C variables to the Promela model.
- c_expr - evaluate a C expression whose return value can be used int the Promela model
- c_code - add arbitrary C code fragments as atomic statements
- c_track - include (external memory into the state vector)

Reachability

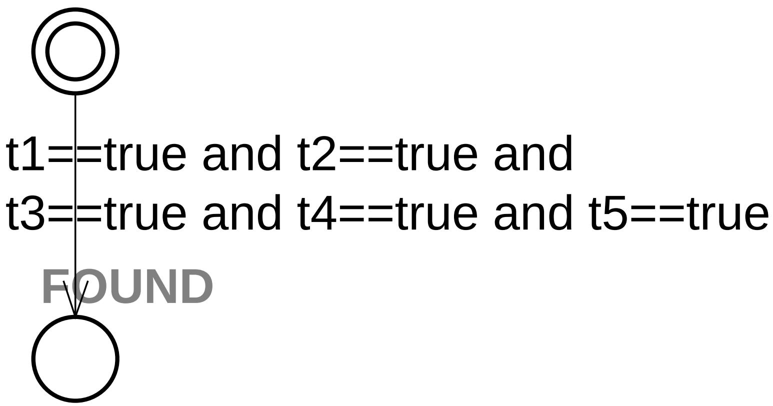
- We formulate the coverage criteria as reachability questions (that can be formulated in terms of , i.e. “there exists a state where some propositional property holds”)
- Many interesting properties can be encoded into reachability problems by means of monitor automata / monitor processes
- As Gordon said, these are the safety properties that can be specified in this way

A Simple Example

System



Goal/monitor



Can this system reach a state where booleans t1, t2, t3, t4 and t5 are true?

Explicit State Model Checking

- We deal with explicit state model checking
  - all control states and data states are represented explicitly.
  - Spin is explicit state; Uppaal is explicit state (except its representation of time)
- As opposed to symbolic model checking
  - where the states are represented by some symbolic construct, for example BDD-s.

Ways of reducing memory consumption

- Partial order reduction
- Symmetry reduction
- Lossless state compression
  - Collapse compression
  - Minimized automaton representation
- Lossy state compression
  - **bit-state hashing**
  - hash compaction

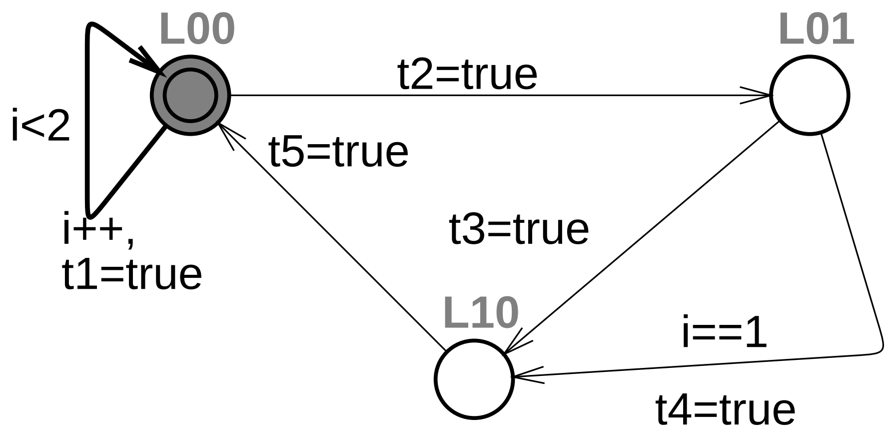
Bit-state hashing

- Let us look at how bit state hashing works.
- Instead of a long string representing a state, store one bit.

$$hash(1000110110110010101010101001) = addr_{bit}$$

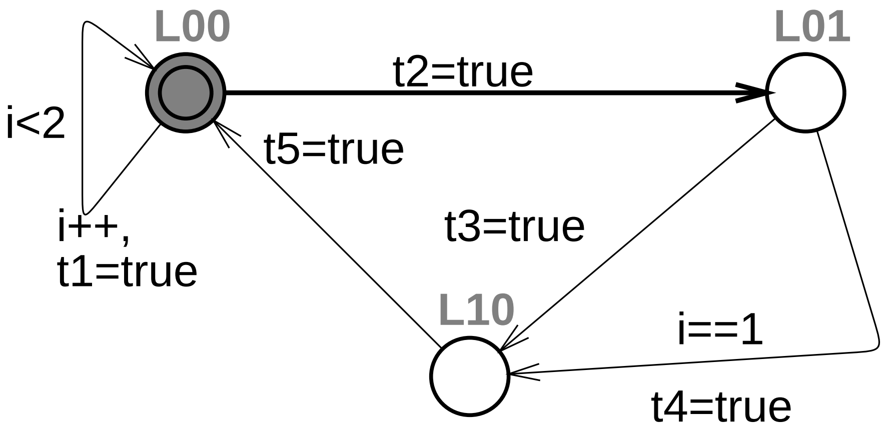
Iterated Search Refinement

- Three states can be encoded as 2 bits
- Each boolean is one bit
- Integer i is in range 0 to 3, thus 2 bits.



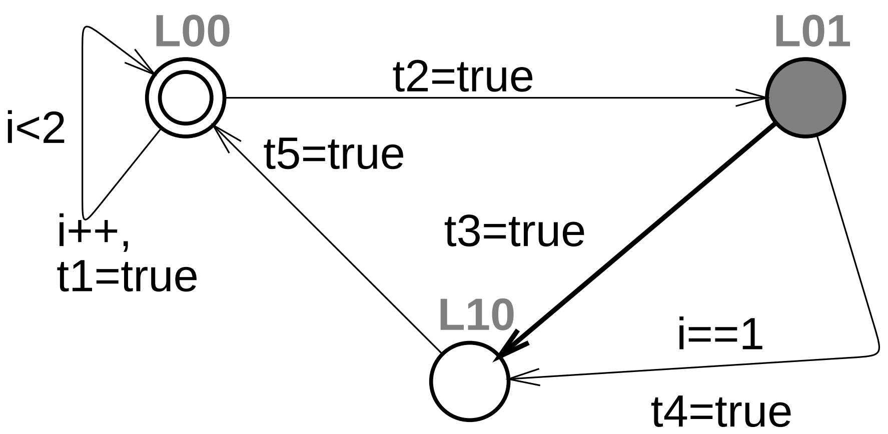
Path	Bit 0 of i	Bit 1 of i	t1	t2	t3	t4	t5	Bit 0 of L	Bit 0 of L	$State_{10}$	mod 9	mod 10	mod 11	mod 12	mod 13	mod 14
Initial state	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T_1	0	1	1	0	0	0	0	0	0	192	3	2	5	0	10	10

Iterated Search Refinement



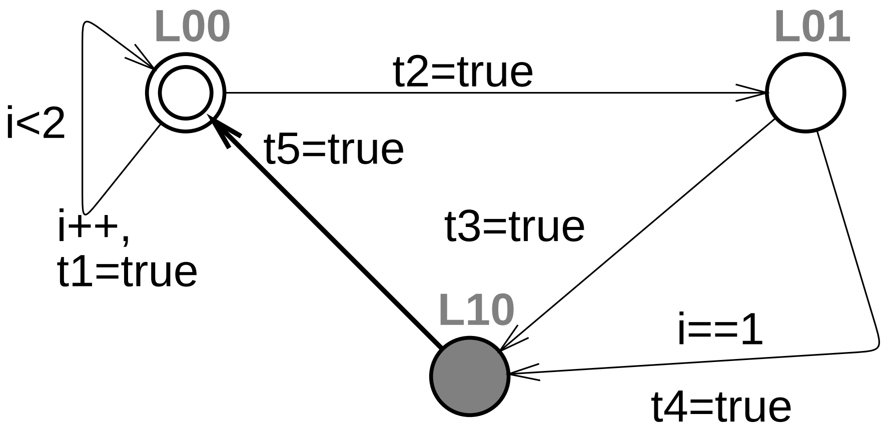
Path	Bit 0 of i	Bit 1 of i	t 1	t 2	t 3	t 4	t 5	Bit 0 of L	Bit 0 of L	$State_{10}$	mod 9	mod 10	mod 11	mod 12	mod 13	mod 14
Initial state	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T_1	0	1	1	0	0	0	0	0	0	192	3	2	5	0	10	10
T_1	1	0	1	0	0	0	0	0	0	320	5	0	1	8	8	12
T_2	1	0	1	1	0	0	0	0	1	353	2	3	1	5	2	3
T_3	1	0	1	1	1	0	0	1	0	370	1	0	7	10	6	6
T_5	1	0	1	1	1	0	1	0	0	372	3	2	9	0	8	8

Iterated Search Refinement



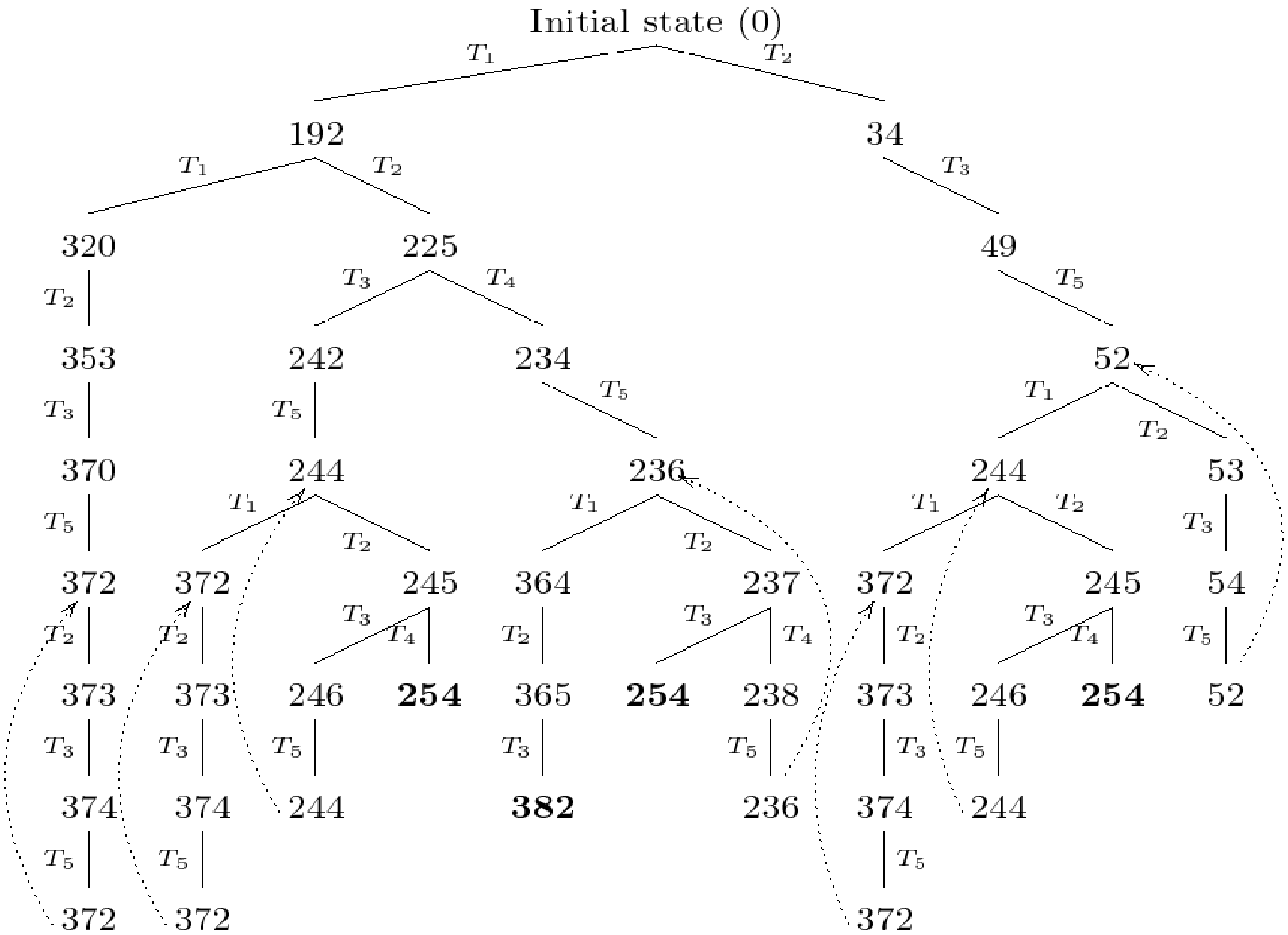
Path	Bit 0 of i	Bit 1 of i	t 1	t 2	t 3	t 4	t 5	Bit 0 of L	Bit 0 of L	$State_{10}$	mod 9	mod 10	mod 11	mod 12	mod 13	mod 14
Initial state	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T_1	0	1	1	0	0	0	0	0	0	192	3	2	5	0	10	10
T_1	1	0	1	0	0	0	0	0	0	320	5	0	1	8	8	12
T_2	1	0	1	1	0	0	0	0	1	353	2	3	1	5	2	3
T_3	1	0	1	1	1	0	0	1	0	370	1	0	7	10	6	6
T_5	1	0	1	1	1	0	1	0	0	372	3	2	9	0	8	8

Iterated Search Refinement

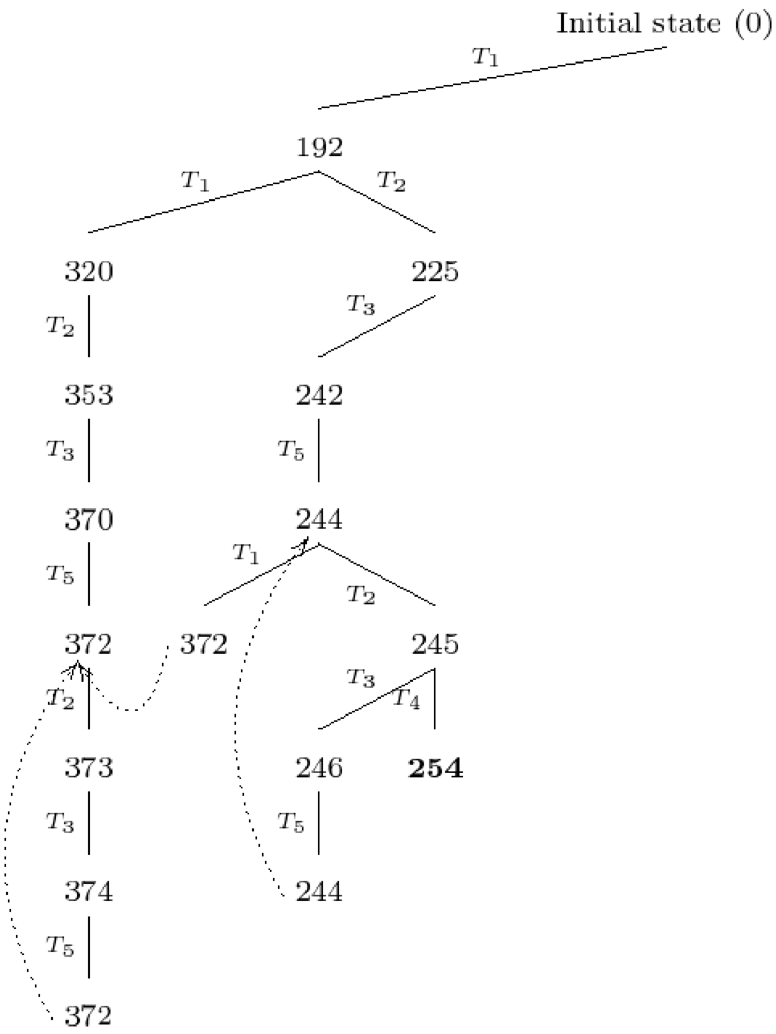


Path	Bit 0 of i	Bit 1 of i	t1	t2	t3	t4	t5	Bit 0 of L	Bit 0 of L	$State_{10}$	mod 9	mod 10	mod 11	mod 12	mod 13	mod 14
Initial state	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T_1	0	1	1	0	0	0	0	0	0	192	3	2	5	0	10	10
T_1	1	0	1	0	0	0	0	0	0	320	5	0	1	8	8	12
T_2	1	0	1	1	0	0	0	0	1	353	2	3	1	5	2	3
T_3	1	0	1	1	1	0	0	1	0	370	1	0	7	10	6	6
T_5	1	0	1	1	1	0	1	0	0	372	3	2	9	0	8	8

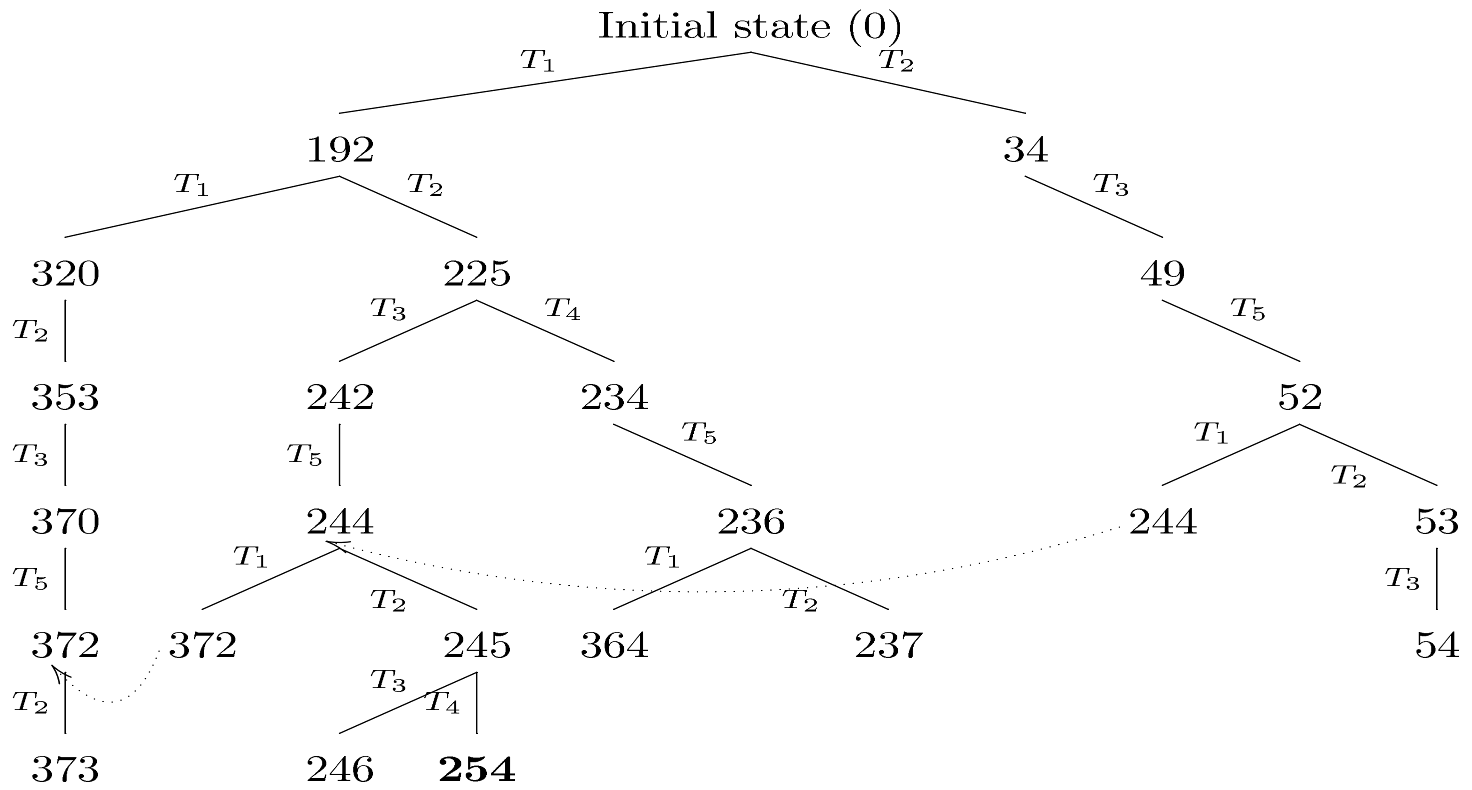
Full search tree



Depth first search tree

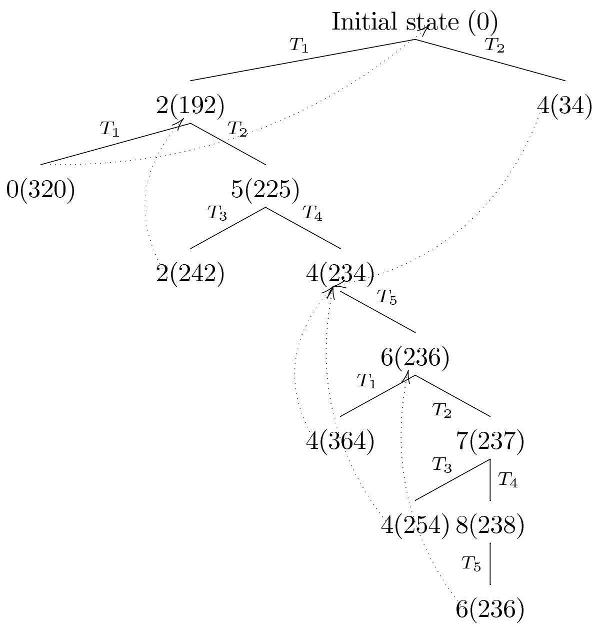


Breadth First Search Tree

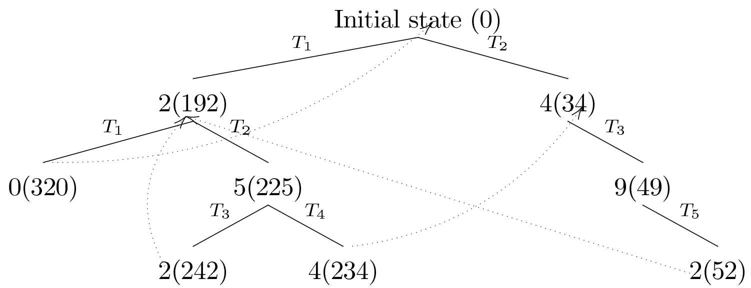


Search Tree mod 10

Depth first

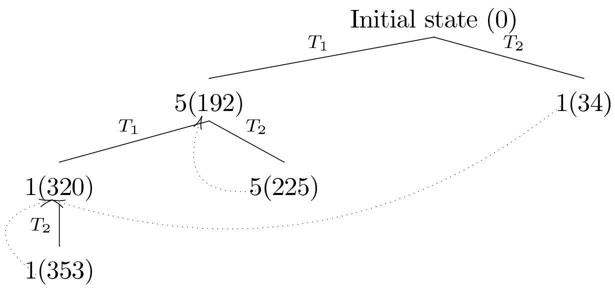


Breadth first

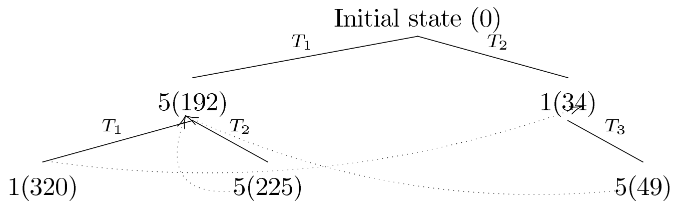


Search Tree mod 11

Depth first

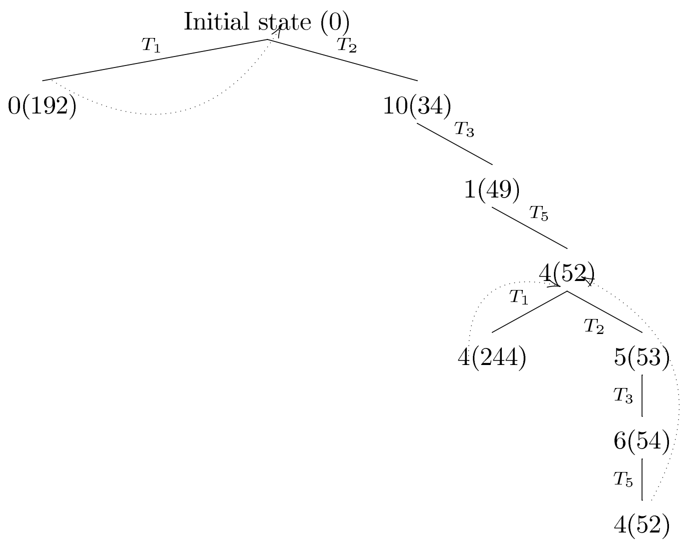


Breadth first

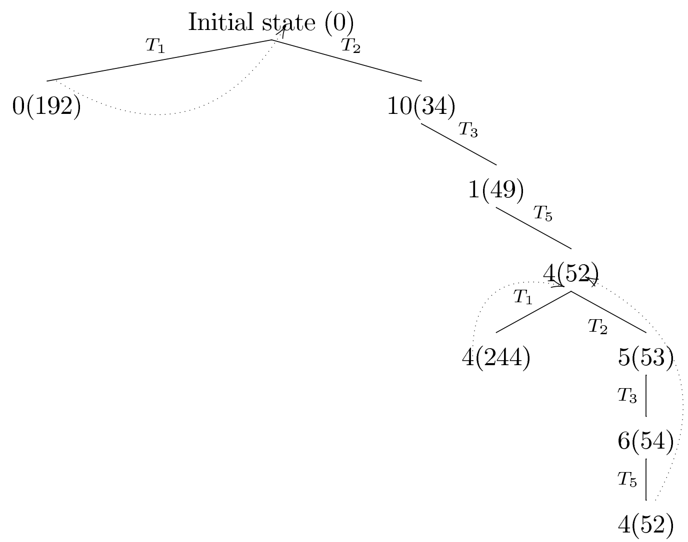


Search Tree mod 12

Depth first



Breadth first



Guiding

- Guiding drives the the model checker in a direction that is not obviously wasteful
- The smarter the guiding the shorter the sooner reasonably good solutions are found and thus search space is pruned

Guided model checking

- There are extended model checkers where it is possible to guide search using a heuristic variable or a cost variable. (e.g. Uppaal-Cora in addition to Spin)
  - (similar to priorities in SyncCharts)
- Even in the guided case, the model checker wants to use far too much memory.
- Using intentionally underdimensioned bit state table sizes yields interesting results!

Conclusion

- Optimal scheduling problems can be specified using Promela and how Spin can be used to find a solution;
- Branch and bound can be implemented in Promela using calls to internal functions of Spin
- Bitstate hashing based iterated search refinement:
  - enables to increase the size of the spec / use more complicated coverage criteria
  - combined with guiding helps to find much shorter test sequences (than with DFS)

Thank you for your attention!

References

- Theo Ruys. Optimal Scheduling Using Branch and Bound with SPIN 4.0. SPIN Workshop 2003
- Juhan P. Ernits, Andres Kull, Kullo Raiend and Jüri Vain. Generating Tests from EFSM Models using Guided Model Checking and Iterated Search Refinement. Proceedings of FATES/RV'06